New Fast and Area-Efficient Pipeline 3-D DCT Architectures

Saad Al-Azawi[a], Omar Nibouche[b], Said Boussakta[c], Gaye Lightbody[d]
[a] College of Engineering, Diyala University, Diyala, Iraq; [b,d] Faculty of Computing and Engineering, Ulster University, UK; [c] School of Engineering, Newcastle University, UK
[a]saad.alazawi@engineering.uodiyala.edu.iq, [b]o.nibouche@ulster.ac.uk, [c]said.boussakta@newcastle.ac.uk, [d]g.lightbody@ulster.ac.uk

Abstract
The efficient implementation of 3-D transforms is a challenging task due to the computation complexity, memory and area requirements of such transforms. One important 3-D transform is the 3-D Discrete Cosine Transform (3-D DCT) used in many image and video processing systems. In this paper, two new pipeline architectures for the 3-D DCT computation using the 3-D DCT Vector-Radix algorithm (3-D DCT VR) are presented. These architectures are scalable and parameterisable with regards to different wordlengths and pipelining levels. Their arithmetic component requirements are reduced to the order of $O(log_2 N)$ in contrast with $O(N)$ for 3-D DCT architectures in the literature, while at the same time they can keep similar or better area-time complexity.

Key words: 3-D Discrete Cosine Transform (DCT), Row-Column (RC), Row-Column-Frame (RCF), Vector-Radix, FPGA

1. Introduction

Transforms such as the Fourier Transform (FT) [1-5], Wavelet Transform (WT) [6-9], and Cosine Transform (CT) [10-14] play a critical part in various Digital Signal Processing (DSP) applications, including audio, image and video systems. Much of the usefulness of these transforms arises from their frequency and time-frequency representations and properties including the decorrelation property, energy compactness, and the availability of fast algorithms for their computation. Nevertheless, even the fast algorithms that implement these transforms are still very computationally intensive. Thus, these transforms can become a bottleneck in terms of the system's speed, and contribute greatly to their area usage and power consumption [1-3, 6-8, 10-19]. For its role in many image and video applications, including the JPEG, MPEGx and H.26x compression standards, the DCT has received a great deal of research interest [20-24]; the 1-D and 2-D DCT are now the established transforms for many applications and standards. Further, there are many new and emerging applications for the 3-D DCT, including visual tracking, video coding and watermarking [25-29].

Numerous 1-D and 2-D DCT architectures have been suggested in the literature [30-39]. Exploiting the separability principle of the transform, 2-D DCT cores based on the 1-D DCT Row-Column (RC) approach are suggested in [33-36]; yet very few architectures that implement the 3-D DCT can be found [38-45]. Traditionally, the 3-D DCT has been implemented by cascading stages of the 1-D DCT as in the well-known Row-Column-Frame (RCF) approach. Noteworthy differences between architectures in the literature are their level of parallelisation in terms of the number of stages and the number of 1-DCT cores per stage, which leads to different trade-offs between circuit complexity and throughput. One common architecture employs three stages of one 1-D DCT core and N^3+N^2–word transpose memory [40-42]. Parallelisation can be applied to the first two 1-D DCT cores which in fact implements a 2-D DCT transform, leading to the utilisation of *2N+1* 1-D DCT processors and N^3+N-word memory [41, 42]. Another class of the 3-D DCT architectures multiplexes the 1-D DCT transforms involved in its computation onto a single 1-D DCT architecture. Such a class of architecture requires N^3-word memory [41, 42]. The reduction achieved in hardware utilisation comes at the cost of a lower throughput. Using three 1-DCT cores to implement the 3-D DCT achieves a throughput three times higher than when employing a single 1-D DCT processor. The throughput is *N*-fold augmented via parallelisation of the 1-D DCT processors [42]. The 1-D DCT cores employed in the 3-D DCT architecture can use the transform's fast algorithm, distributed arithmetic or ROM based designs [38]. Such architectures exhibit irregular structures, lack of modularity, and complex control. Another class of the 3-D DCT architecture relies solely on the systolic approach with its well established design methodology [43]. In [44, 46], high speed and low complexity pipeline n-D DCT architectures are proposed using the regular 1-D DCT and tensor product operations. The proposed architectures are based on the 1-D and 2-D DCT architectures in [47].

In this paper, two new pipeline and scalable architectures that implement the 3-D Discrete Cosine Transform Vector-Radix (3-D DCT VR) are introduced. The presented architectures are parameterisable in terms of wordlength and pipeline stages. Further, no block memory is used for data transposition. These architectures have been implemented and tested; for instance, an FPGA-based implementation of a 512×512×8-word data using a transform length of 8×8×8-word cube size and a 14-bit wordlength has achieved a working frequency of 330 MHz and a processing time of 6.4 ms. Thus, 80000 frames can be processed in every second.

The remainder of this paper is organised as follows. In section 2, the background of the DCT transform and 3-D DCT VR algorithm are provided. Sections 3, 4 and 5 present the two new architectures for 3-D DCT computation. The results

obtained are discussed in section 6 and conclusions are given in section 7.

2. Background and the 3-D DCT VR Algorithm

The 3-D DCT coefficients of a *N×N×N* data cube are computed as follows:

$$X(k_1,k_2,k_3) = \frac{8\varepsilon_{k1}\varepsilon_{k2}\varepsilon_{k3}}{N^3}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\sum_{n_3=0}^{N-1} x(n_1,n_2,n_3)\ \cos\left(\frac{\pi}{2N}(2n_1+1)k_1\right)\cos\left(\frac{\pi}{2N}(2n_2+1)k_2\right)\cos\left(\frac{\pi}{2N}(2n_3+1)k_3\right) \quad (1)$$

where k_i and n_i =0, 1, 2, …. *N-1*, *i*=1,2,3 and

$$\varepsilon_{ki} = \begin{cases} \frac{1}{\sqrt{2}}, & for\ k_i = 0 \\ 1, & otherwise \end{cases}$$

Equation (1) represents the definition of the 3-D DCT; as the 3-D DCT is a separable transform, it can be computed by applying the *N*-point 1-D DCT algorithm to the row, column and frame directions [25]. As such, the computation complexity of the 3-D DCT RCF algorithm is $3N^2$-time that of the 1-D DCT algorithm. Further, it requires $\frac{3}{2}N^3 log_2 N$ multiplication and $\frac{9}{2}N^3 log_2 N - 3N^3 + 3N^2$ addition operations [25, 48, 49]. However, it has been shown that a further reduction of the computation requirement can be achieved by using the 3-D DCT VR referred to as 3-D DCT-II VR [25, 48]. With such a VR algorithm, a saving of more than 40% of the total number of multiplication operations is achieved the number of multiplication operations is reduced to $\frac{7}{8}N^3 log_2 N$ operations while the number of additions is kept the same when compared with the familiar RCF approach.

The 3-D DCT VR algorithm includes four computation steps; namely data reordering, a butterfly calculation unit that comprises $log_2 N$ butterfly stages, bit-reverse ordering and post addition, as illustrated in Figure 1. The algorithm partitions the input into cubes of *N×N×N* points, where *N* is a power of two. Each data cube is rearranged according to the index mapping of equation (2) as follows:

$$\begin{bmatrix} \tilde{x}(n_1,n_2,n_3) \\ \tilde{x}(n_1,n_2,N-n_3-1) \\ \tilde{x}(n_1,N-n_2-1,n_3) \\ \tilde{x}(n_1,N-n_2-1,N-n_3-1) \\ \tilde{x}(N-n_1-1,n_2,n_3) \\ \tilde{x}(N-n_1-1,n_2,N-n_3-1) \\ \tilde{x}(N-n_1-1,N-n_2-1,n_3) \\ \tilde{x}(N-n_1-1,N-n_2-1,N-n_3-1) \end{bmatrix} = \begin{bmatrix} x(2n_1,2n_2,2n_3) \\ x(2n_1,2n_2,2n_3+1) \\ x(2n_1,2n_2+1,2n_3) \\ x(2n_1,2n_2+1,2n_3+1) \\ x(2n_1+1,n_2,n_3) \\ x(2n_1+1,n_2,2n_3+1) \\ x(2n_1+1,2n_2+1,n_3) \\ x(2n_1+1,2n_2+1,2n_3+1) \end{bmatrix} \quad (2)$$

where $n_i = 0, 1, \ldots, \frac{N}{2} - 1$, *i=1, 2,3*, and signal $\tilde{x}$ is the reordered version of the original input x.

The subsequent computation stage of the 3-D DCT VR algorithm is the butterfly computation. The reordered data $\tilde{x}$ is inserted in (1) to produce:

$$X(k_1,k_2,k_3) = \frac{8\varepsilon_{k1}\varepsilon_{k2}\varepsilon_{k3}}{N^3}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\sum_{n_3=0}^{N-1}\tilde{x}(n_1,n_2,n_3) \times \cos(\emptyset_1 k_1)\cos(\emptyset_2 k_2)\cos(\emptyset_3 k_3) \quad (3)$$

where $\emptyset_i = \frac{\pi}{2N}(4n_i+1)$ and *i*=1,2,3.

By considering even and odd indices, the 3-D DCT can be computed as follows:

$$X(2k_1+i,2k_2+j,2k_3+l) = \sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}\left[\tilde{x}_{ijl}(n_1,n_2,n_3)\right] \times cos\left(\emptyset_1(2k_1+i)\right)cos\left(\emptyset_2(2k_1+j)\right)cos\left(\emptyset_3(2k_1+l)\right) \quad (4)$$

where *ijl=[000,001,010,011,100,101,110,111]*, $M = {}^{N}/_{2} - 1$ and:

$$\tilde{x}_{ijl}(n_1,n_2,n_3) = \tilde{x}(n_1,n_2,n_3) + (-1)^l\tilde{x}\left(n_1,n_2,n_3+\frac{n}{2}\right) + (-1)^j\tilde{x}\left(n_1,n_2+\frac{n}{2},n_3\right) + (-1)^{j+l}\tilde{x}\left(n_1,n_2+\frac{n}{2},n_3+\frac{n}{2}\right) + (-1)^i\tilde{x}\left(n_1+\frac{n}{2},n_2,n_3\right) + (-1)^{i+l}\tilde{x}\left(n_1+\frac{n}{2},n_2,n_3+\frac{n}{2}\right) + (-1)^{i+j}\tilde{x}\left(n_1+\frac{n}{2},n_2+\frac{n}{2},n_3\right) + (-1)^{i+j+l}\tilde{x}\left(n_1+\frac{n}{2},n_2+\frac{n}{2},n_3+\frac{n}{2}\right) \quad (5)$$

If k_1, k_2 and k_3 are even then (5) can be rewritten as:

$$X(2k_1,2k_2,2k_3) = \left\{ \sum_{n_1=0}^{M}\sum_{n_2=0}^{M}\sum_{n_3=0}^{M}\left[\tilde{x}_{000}(n_1,n_2,n_3)\right] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} \quad (6)$$

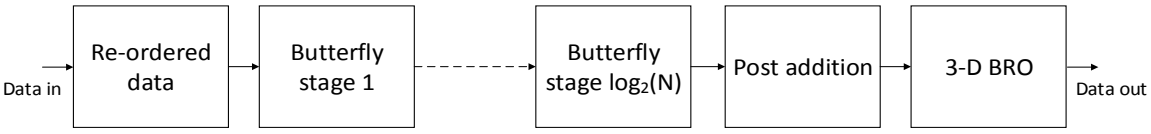


Figure 1. Block diagram of the 3-D DCT VR algorithm.

For the remaining combinations of odd/even indices k_1, k_2 and k_3, one can divide the computation of the transform as follows:

$$X(2k_1, 2k_2, 2k_3+1) = \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [2\tilde{x}_{001}(n_1, n_2, n_3) \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1, 2k_2, 2k_3 - 1) \tag{7}$$

$$X(2k_1, 2k_2+1, 2k_3) = \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [2\tilde{x}_{010}(n_1, n_2, n_3) \cos \emptyset_2] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1, 2k_2 - 1, 2k_3) \tag{8}$$

$$X(2k_1+1, 2k_2, 2k_3) = \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [2\tilde{x}_{100}(n_1, n_2, n_3) \cos \emptyset_1] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1 - 1, 2k_2, 2k_3) \tag{9}$$

$$\begin{aligned} &X(2k_1, 2k_2+1, 2k_3+1) \\ &= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{011}(n_1, n_2, n_3) \cos \emptyset_2 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1, 2k_2 - 1, 2k_3 + 1) \\ &- X(2k_1, 2k_2 + 1, 2k_3 - 1) - X(2k_1, 2k_2 - 1, 2k_3 - 1) \end{aligned} \tag{10}$$

$$\begin{aligned} &X(2k_1+1, 2k_2, 2k_3+1) \\ &= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{101}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1 - 1, 2k_2, 2k_3 + 1) \\ &- X(2k_1 + 1, 2k_2, 2k_3 - 1) - X(2k_1 - 1, 2k_2, 2k_3 - 1) \end{aligned} \tag{11}$$

$$\begin{aligned} &X(2k_1+1, 2k_2+1, 2k_3) \\ &= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [4\tilde{x}_{110}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_2] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} - X(2k_1 - 1, 2k_2 + 1, 2k_3) \\ &- X(2k_1 + 1, 2k_2 - 1, 2k_3) - X(2k_1 - 1, 2k_2 - 1, 2k_3) \end{aligned} \tag{12}$$

$$\begin{aligned} &X(2k_1+1, 2k_2+1, 2k_3+1) \\ &= \left\{ \sum_{n_1=0}^{M} \sum_{n_2=0}^{M} \sum_{n_3=0}^{M} [8\tilde{x}_{111}(n_1, n_2, n_3) \cos \emptyset_1 \cos \emptyset_2 \cos \emptyset_3] \times \prod_{i=1}^{3} cos(2\emptyset_i k_i) \right\} \\ &- X(2k_1 + 1, 2k_2 + 1, 2k_3 - 1) - X(2k_1 + 1, 2k_2 - 1, 2k_3 + 1) - X(2k_1 + 1, 2k_2 - 1, 2k_3 - 1) \\ &- X(2k_1 - 1, 2k_2 + 1, 2k_3 + 1) - X(2k_1 - 1, 2k_2 + 1, 2k_3 - 1) - X(2k_1 - 1, 2k_2 - 1, 2k_3 + 1) \\ &- X(2k_1 - 1, 2k_2 - 1, 2k_3 - 1) \end{aligned} \tag{13}$$

The set of equations (6)-(13) represents a single butterfly computation of a Decimation In Frequency (DIF) VR algorithm as shown in Figure 2. It computes eight points; a butterfly can receive a *N×N×N* data cube at its input and outputs 8 data cubes of *N/2×N/2×N/2*-word each; the process can be repeated until $\frac{N^3}{8}$ data cubes of *2× 2× 2*-word each are computed. Thus, the flow graph of the whole butterfly computation consists of $log_2 N$ stages with $\frac{N^3}{8}$ butterflies per stage [25]. The output from the last butterfly stage is fed to the post addition stages then it is bit-reversed. The post addition operations, shown by the terms outside braces in the set of equations (6)-(13), are then carried out. Further to the

reduction in arithmetic complexity and processing time, the 3-D DCT VR algorithm does not require transpose memory and exhibits a regular butterfly structure which is more suitable for hardware and software implementation than the RCF approach. Further details about the 3-D DCT VR algorithm can be found in [25].

The presented architectures both partition the input sequence into cubes of *N*×*N*×*N*-word or *N*-blocks of *N*×*N*-word. The input data is reordered according to (2). The reordering process is performed by shuffling words along the row, column and frame dimensions. It includes dividing data into odd-indexed and even-indexed words and retrograde indexing. As

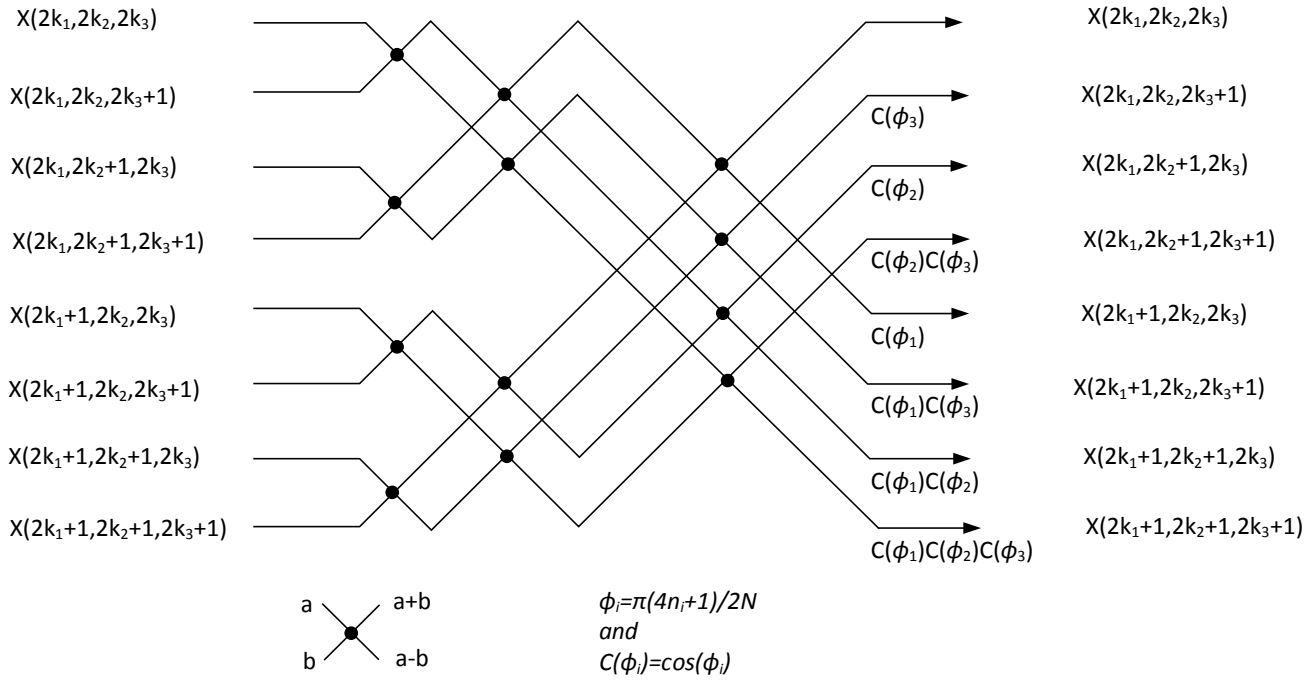


Figure 2. Single butterfly of the 3-D DCT DIF VR algorithm

3. New 3-D DCT Vector Radix Architectures

Two new architectures are presented; namely the Single Path Data Flow 3-D DCT Architecture (SPDFA) and the Dual Path Data Flow 3-D DCT Architecture (DPDFA). The difference between them lies in the number of words fed to the adjacent butterfly and how the arithmetic operations are scheduled within each butterfly stage; this has led to the derivation of two structures with different hardware requirements. Both architectures are built according to the generic block diagram of Figure 1. The butterfly calculation consists of $log_2 N$ parameterised and scalable stages as described by the set of equations (6)-(13) and illustrated in Figure 2. The data reordering is common to both SPDFA and DPDFA, however, the internal architecture of the butterfly, post-addition stages and the 3-D Bit Reverse Ordering (3-D BRO) stage are architecture-dependent. Of the two presented structures, SPDFA exhibits a single line of data between neighbouring butterfly stages. It is more efficient in memory usage as intermediate results are fed back to memory elements; however, using these feedback loops prevents any further pipelining. DPDFA is a dual-path data flow feed-forward architecture. There are two data lines between adjacent butterflies and further pipelining is a simple task; the architecture however requires more memory than SPDFA.

an example, for *N* indices arranged as "*0, 1, 2, 3, 4, 5, 6, … N-1"*, the reordered sequence will be "*0, 2, 4, 6, …, N-2, N-1, N-3, N-5, ….., 1*". This stage is implemented using a dual port block RAM which permits writing and reading operations to be performed on different locations during the same cycle. Thus, for an *N*×*N*×*N*-word cube, the memory size required for the reordering operation is $\left(\frac{N}{2}+1\right)N^2$–word with a latency of $\frac{N^3}{2}$ cycles as only writing operations are carried out during this period.

4. Single Path Data Flow 3-D DCT Architecture

SPDFA is composed of a 3-D reordering stage, $log_2(N)$ butterfly computation stages, three post addition sub-stages and a 3-D Bit Reverse Order (3-D BRO) stage.

4.1 Butterfly Stages

The reordered data from the 3-D reordering stage is fed to the butterfly stages at a rate of one word per clock cycle. As shown in Figure 3, $log_2 N$ butterfly stages (*m=1, 2, 3,.., $log_2 N$*) are used. Each butterfly stage can be further divided into three sub-stages and a multiplier as shown in Figure 4. A sub-stage consists of two add/subtract elements for carrying out addition and subtraction operations between the two halves of each input along the three dimensions of data, a register and a switch. The multiplier is used to multiply the output

words by appropriate Twiddle Factors (TFs) which are pre-computed and stored in a look up table (LUT).

For the sake of explaining, the words $x(n_1, n_2, n_3)$ at the input of the first butterfly stage can be indexed as $x(n_1 + n_2 \times N + n_3 \times N^2)$. The first sub-stage performs addition and subtraction between the two halves of the input data cube; the first half contains words indexed from 0 to $\frac{N^3}{2} - 1$ and the second part the words with indices from $\frac{N^3}{2}$ $to\ N^3 - 1$. During the first $\frac{N^3}{2}$ clock cycles, the first part of the data is stored in *Register 1* (of length $\frac{N^3}{2}$-word) before being fed to adders along with the input data from the second half during the next $\frac{N^3}{2}$ cycles. During this period, the subtraction operation results are stored in *Register 1* while the addition results are fed to the next sub stage. Once this is completed, it is the turn of the subtraction results stored in *Register 1* to be fed to the next sub-stage. The selection of which part of the data to be stored in *Register 1*, fed to the adders or fed to the next sub-stage is managed by the control signal of *Switch 1,* which changes its value every $\frac{N^3}{2}$ cycles.

What the first sub-stage carries out on cubes of data, the second sub-stage performs it on blocks of *N*×*N*-word of the same data cube. Omitting changes along n_3, each block is again divided into two halves; one half includes indices $n_1 + n_2 \times N$ from $0\ to\ \frac{N^2}{2} - 1$ while the second half includes words of the same block with indices from $\frac{N^2}{2}$ $to\ N^2 - 1$. The behaviour of the second sub-stage is similar to the first one; except that *Register 2* is of length $\frac{N^2}{2}$-word and the period of the control signal for *Switch 2* is N^2 cycles with a duty cycle of 50%.

The third sub-stage implements addition and subtraction between the two halves of each column in each block using *Register 3 (*of length $\frac{N}{2}$-word). Omitting changes of n_3 and n_2, the data in each column is divided into two halves with indices ranging from $0\ to\ \frac{N}{2} - 1$ and from $\frac{N}{2}$ $to\ N - 1$. The words of the first half of the column are stored in *Register 3*, they are then fed to the adders along with the column's second half. The results of the addition operation are multiplied by the appropriate TFs. After which, the results of the subtraction operation which were first stored in *Register 3* are fed to the multiplier for the multiplication by the TFs. The multiplier output is input to the next butterfly stage. As with sub-stages 1 and 2, *Switch 3* multiplexes data and its control signal is periodic and changes its value every $\frac{N}{2}$ cycles.

In the general case of the m^{th} butterfly stage, data is split into 2^{3m-3} cubes of $\left(\frac{N}{2^{m-1}}\right)^3$ words; the first butterfly sub-stage is used to perform the addition and subtraction operations between the two halves of each input data cube; the words involved are indexed from 0 to $\frac{N^3}{2^m} - 1$ and from $\frac{N^3}{2^m}$ to $\frac{N^3}{2^{m-1}}$. During the first $\frac{N^3}{2^m}$ cycles, the data cube's first half is stored in *Register 1*; in the next $\frac{N^3}{2^m}$ cycles, the addition and subtraction operations take place; the results of the addition operation are fed to the adjacent butterfly sub-stage while the results of the subtraction operations are stored in *Register 1* before being fed to the adjacent sub-stage in the next $\frac{N^3}{2^m}$ cycles. In a similar way and with an appropriate switching, the second and third butterfly sub-stages implement the addition and subtraction operations between the first and second halves of the data blocks and columns, respectively. The lengths of registers, *Register 2* and *Register 3,* is $\frac{N^2}{2^m}$-word and $\frac{N}{2^m}$-word, respectively, which allows for storing half of each block and column of data as appropriate. The multiplication operation by a twiddle factor (TF) takes place once all arithmetic operations of sub-stage 3 have been carried out.

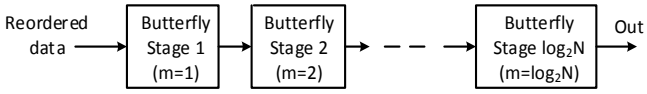


Figure 3. The block diagram of butterfly stages

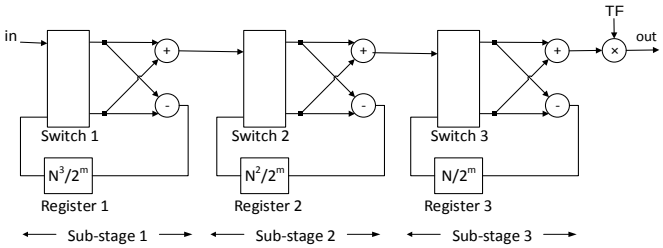


Figure 4. SPDFA m^{th} butterfly internal architecture

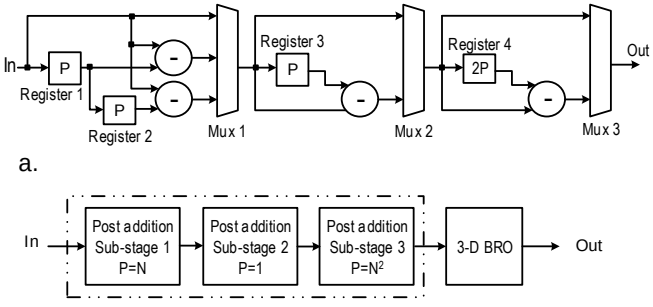


Figure 5. a. A parameterised post addition sub- stage for SPDFA. b. A post addition stage and 3-D BRO

4.2 Post Addition and 3-D BRO Stages

The third part of SPDFA is the post addition stage which performs the computation of the terms outside the curly brackets in (6)-(13). Reflecting the three dimensions of the input data, the post addition stage can be divided into three sub-stages; each sub-stage carries out addition operations over a given dimension. In the first, second and third post addition sub-stage, the addition operations are carried out within the same *N*×*N*-word block, the same column or the

same data cube; respectively. Hence, the length of the registers, used in Figure 5 and labelled as parameter P, may vary. Still the internal architecture of each sub-stage is identically the same.

The output of the third post addition stage is fed to the 3-D BRO stage which performs data reordering as the fast algorithm used introduces a bit reversal permutation on the binary indices of the results. Bit reversal is performed along each row, column and frame directions in each *N×N×N*-word data cube using a regular bit reversal algorithm [25]. The output from this stage represents the 3-D DCT coefficients of the input data. It is implemented using a $\left(\frac{3N}{4}-1\right)N^2$-word dual-port block RAM. This stage is placed next to the post addition stage to act as a buffer for the subsequent system if required; for instance, it can be integrated with a quantizer as in conventional data compression algorithms.

5. Dual Path Data Flow 3-D DCT Architecture

DPDFA is a dual data path architecture for the 3-D DCT VR computation. It is devised to produce a high speed 3-D DCT architecture which can be easily retimed and pipelined. DPDFA consists of 3-D data reordering, butterfly stages, post addition and 3-D BRO stages. The 3-D data reordering stage is the same as that presented earlier in the paper.

5.1 Butterfly Stages

The scheduling of arithmetic operations in DPFDA is different from SPDFA. Rather than feeding the subtraction operations intermediate results back to the same sub-stage register as in SPDFA, the results of the addition and subtraction are fed forward to the next sub-stage or to the next stage. This reduces the time during which registers are utilised for storing partial results, adds another line of data for communication between adjacent stages and sub-stages, and hence increases the number of required multipliers to cope with the computation of two coefficients per clock cycle. However, this simplifies pipelining and retiming in the DPDFA.

DPDFA comprises $log_2 N$ butterfly stages; each can be divided into three sub-stages, registers, switches and two multipliers. A generic sub-stage consists of two add/subtract elements for carrying out addition and subtraction operations between the two halves of each input along the three dimensions of data. It also contains two registers and a switch for data ordering and multiplexing; the exception is the first sub-stage of the first butterfly which utilises only one register as shown in Figure 6. The first butterfly internal architecture takes into account the fact that data is received at its input at the rate of one word per clock cycle which are then stored and processed at the rate of two words per clock cycle.

The first sub-stage performs addition and subtraction between the two halves of the input data cube; *Register 1* stores the first half that contains words of indices from 0 to $\frac{N^3}{2}-1$ then feeds it to the adder components during the next $\frac{N^3}{2}$ cycles when the second data cube part that contains words indexed from $\frac{N^3}{2}$ to N^3-1 is also available at the input of the adder components. Data multiplexing is carried out using *Switch 1* which is used to twofold parallelise the serial input. Its control signal is periodic with a period of N^3 cycles and a 50% duty cycle.

In sub-stage 2, the registers *Register 2* and *Register 3*, and *Switch 2* re-order data with the aim to implement the addition and subtraction operations in each block of data; a block is divided into two halves; words with indices from 0 to $\frac{N^2}{2}-1$ are stored in *Register 3* while the second half which includes words of the same block with indices from $\frac{N^2}{2}$ *to* N^2-1 is stored in *Register 2*. The flow of data between sub-stages 1 and 2 and the selection of where and when results are stored in registers *Register 2* and *Register 3* is carried out by *Switch 2*. Such a switch has a 50% duty cycle control signal with a period of N^2 cycles.

When sub-stage 2 processes blocks of data of the same cube, in a similar way the third stage carries out the addition and subtraction operations on columns of data belonging to the same block of data. For $N/2$ cycles, the addition results of sub-stage 2 are fed to *Register 5*; the subtraction operation results stored in *Register 4* are fed to the adder components of sub-stage 3; during the next $N/2$ cycles, *Register 4* is connected to *Register 5* while the results of the addition

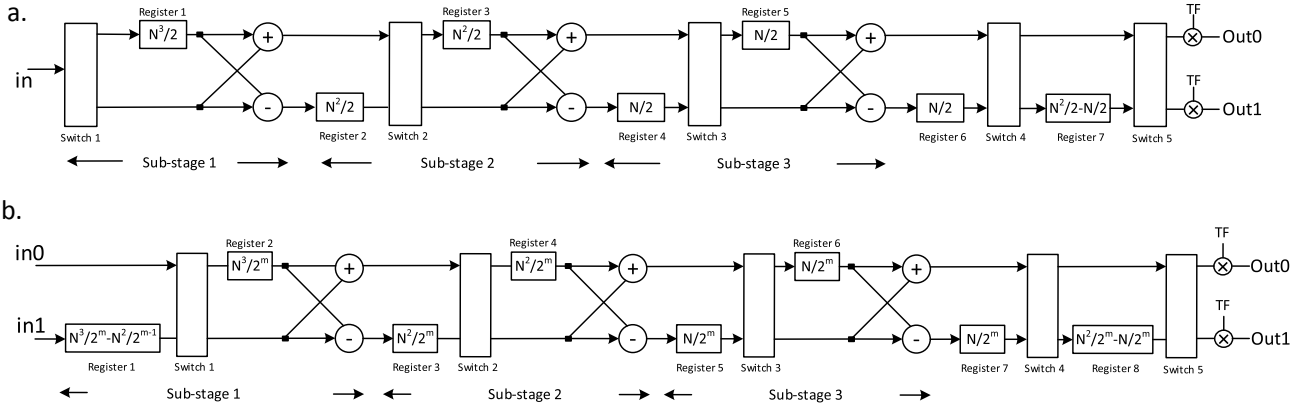


Figure 6. a. The first butterfly of DPDFA, b. The m^{th} butterfly of DPDFA

operation of sub-stage 2 are fed to the adder components of sub-stage 3. Both registers *Register 4* and *Register 5* are of a length of $N/2$-word. *Switch 3* which allows for data switching and controls the flow of partial results in sub-stage 3 has a periodic control signal which changes its value every $N/2$ cycles. Once all addition and subtraction operations have been carried out by the first butterfly three sub-stages, two further tasks have to be carried out, namely; the multiplication by the appropriate TFs and re-arranging data in an order suitable for the next butterfly stage operations to be executed.

Re-arranging data in SPDFA butterflies is simply carried out by feedback registers. However, to re-arrange data in DPSFA one has to cancel out the data order engendered by the selection and switching behaviour of *Switch 2, Switch 3, Register 2, Register 3, Register 4* and *Register 5*. The design approach adopted in this work is to use the same set-up of registers and switches to re-arrange the order of data and then to retime for memory optimization. Hence, the behaviour of *Switch 4* and *Switch 5* is similar to that of *Switch 2* and *Switch 3*, respectively. The impact of using retiming is shown in the length of *Register 7* of the first butterfly of Figure 6.a and in the length of *Register 1* in the first sub-stage of the second butterfly stage illustrated in Figure 6.b. Hence the order of data when it leaves the first butterfly is similar to its order at the adder elements of the first sub-stage.

In the general case, the two data inputs presented at the m^{th} butterfly stage are the two halves of the N^3-word cube; however each half data is ordered as 2^{m-2} sets of $2^{m-1} \times 2^{m-1}$ interleaved data cubes of size $\left(\frac{\mathrm{N}}{2^{\mathrm{m-1}}}\right)^3$ words. The control signals of all switches in Figure 6.b are periodic with a 50% duty cycle. The control signal period of *Switch 1*, *Switch 2* and *Switch 3* is $\frac{N^3}{2^{m-1}}$ cycles, $\frac{N^2}{2^{m-1}}$ cycles and $\frac{N}{2^{m-1}}$ cycles, respectively. By carefully controlling the flow of partial results into registers *Register 1*, *Register 2*, *Register 3*, *Register 4*, *Register 5* and *Register 6* in Figure 6.b, all the addition and subtractions operations can be carried out along the three dimensions of the data. Switches *Switch 4* and *Switch 5,* share the control signals of *Switch 3* and *Switch 2*, respectively. Their switching behaviour and the use of registers *Register 7* and *Register 8,* re-arrange data to the same order it was received at the input of the adder elements of sub-stage 1; the multiplication operations can then take place.

5.2 Post addition Stage and 3-D BRO Stages

The post addition stage can be divided into three sub-stages. To cope with processing two words per clock cycle, the first two sub-stages are built using two of the sub-stages shown in Figure 5~~Figure 5~~.a; the third sub-stage is depicted in Figure 7.a and is composed of five add/subtract elements, registers and a multiplexer. The post addition sub-stages are parameterised. The parameter P shown in Figure 7, refers to the length of registers used.

As with SPDFA, the 3-D BRO stage is required to re-order the output; it adjusts for the bit reversal permutation engendered by the fast transform algorithm used. The 3-D BRO is implemented using $\left(\frac{N}{2}-1\right)N^2$-word dual port block RAM. This memory element can be merged with systems where the presented 3-D DCT core is used.

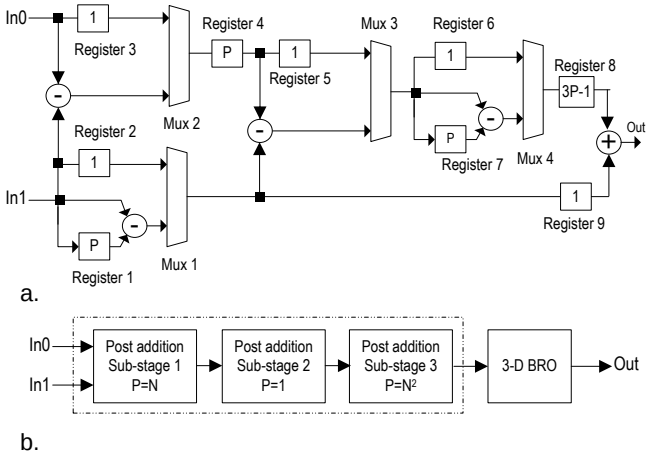


Figure 7: a. Third post addition sub-stage for DPDFA. b. A post addition stage and 3-D BRO for DPDFA.

6. Performance Evaluation

The presented architectures have been designed using Xilinx System generator tool and they have been tested and implemented on a Xilinx Virtex5 5vlx50tff1136-3 FPGA device. Various video sequences and wordlengths have been used to test and evaluate the presented architectures performance and attributes.

6.1 Test Bench and Rate Distortion Performance

DCT$_A$ represents the 3-D DCT of each frame computed using the presented architectures; as they implement the same algorithm, both structures exhibit virtually the same accuracy, as shown in Table 1 and Table 2. When employing DCT_A, the annotation (x,y) refers to a fixed-point wordlength of x+y-bits where x and y represent the number of bits of the integer and fractional parts, respectively. Results of *DCT$_A$* implementation using (12, 8), (12, 4) and (12, 2)-bit wordlengths are shown in this section. In comparison, *DCT$_M$* represents coefficients calculated using Matlab code that implements the 3-D DCT, and $IDCT_M$ represents a Matlab implementation of the inverse 3-D DCT. Both $IDCT_M$ and DCT_M Matlab implementations are floating-point based. For testing and validation purposes DCT_M and *DCT$_A$* have been applied on various MRI and video sequences of $512 \times 512 \times 8$-word [50]; the $IDCT_M$ is then applied to the output of DCT_A to yield reconstructed frames. The peak signal-to-noise ratio (PSNR) and root mean square error (RMSE) are used for evaluating the accuracy of the presented architectures output. The RMSE between the original and reconstructed frames is defined as:

$$RMSE(k) = \sqrt{\frac{1}{P \times Q}\sum_{j=1}^{Q}\sum_{i=1}^{P}\left(IDCT_M\left(DCT_A(i,j,k)\right) - I(i,j,k)\right)^2} \quad (14)$$

Where $I(i,j,k)$ is the original frame and k in the range $1 \leq k \leq F$ is the frame index. *F* is the number of frames in I, P and Q are the number of its rows and columns, respectively. In addition, the PSNR between the original and reconstructed frames is computed as follows [51]:

$$PSNR(k) = 10log\left(\frac{Max(I_k)}{RMSE(k)}\right)^2 \quad (15)$$

where $Max(I_k)$ represents the maximum intensity value of the k^{th} frame. Further, the average of maximum absolute error (AvgMaxErr) of the coefficients for the presented 3-D DCT architectures DCT_A in comparison to the Matlab implementation DCT_M is computed as:

$$MaxErr(k) = Max\left(abs\left(DCT_M(i,j,k) - DCT_A(i,j,k)\right)\right) \quad (16)$$

$$\text{AvgMaxErr} = \frac{1}{F}\sum_{k=1}^{F} MaxErr(k) \quad (17)$$

where $MaxErr(k)$ represents the maximum absolute error for each frame (k).

Performance accuracy, for both presented architectures, was studied over a selection of implementation wordlengths. Table ~~Table~~ 1 and Table ~~Table~~ 2 show that the PSNR increases when the fractional part increases for both SPDFA and DPDFA, respectively. As such and as expected, the highest accuracy is obtained using a 20-bit wordlength (namely, (12, 8)-bit), providing perfect accuracy. The presented architectures produce very good image quality using all the selected wordlengths. The average PSNR of the eight test sequences for SPDFA are ∞, 57 and 45 dB using (12, 8), (12, 4) and (12, 2)-bit wordlengths, respectively. DPDFA achieves very comparable results. Further, in Table ~~Table~~ 1 and Table ~~Table~~ 2, the AvgMaxErr of both architectures are almost identical. For the aim of using visual inspection as a subjective fidelity criterion [52], the images of the original and reconstructed MRI2 scans are shown in Figure 8~~Figure 8~~. For both architectures, the reconstructed images are computed using wordlengths (12,2), (12,4) and (12,8) bits. It can be noticed that the (12,2)-bit wordlength produced a good quality image where the visual error can hardly be noticed; longer wordlengths however lead to a much better quality.

6.2 Area Usage and Computation Time

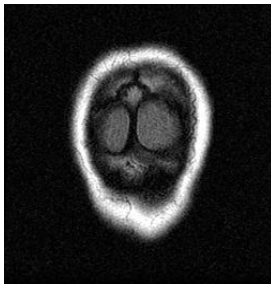
The hardware usage, speed and computation time of both architectures using different wordlengths are shown in Table 3. It is important that the presented architectures are efficient in terms of area usage; in particular, in resource-limited devices such as FPGAs. As it can be seen from Table 3, the average device resources usage of SPDFA and DPDFA is as low as 12% and 18%, respectively. The hardware usage of DPDFA is higher than that of the SPDFA due to duplicate circuitry for multiplication, addition and post addition stages. However, this extra hardware usage and the fact that it has no feedback loops improve the maximum operating frequency of DPDFA over SPDFA. It is easier to place and route the components of DPDFA, including the FPGA device specific resources such as the DSP elements for the implementation of arithmetic operations. As such, the computation time of 512×512×8-word in DPDFA is shorter than that of SPDFA. It is worth pointing out that the memory requirements of both architectures are low in comparison with other architectures due to the in-place computation and the low memory requirement for the BRO and 3-D reordering operations. The memory elements of $5N^2$ and $3N^2$-word have been used for 3-BRO for SPDFA and DPDFA respectively. Further, a memory of $5N^2$-word for 3-D reordering operation has been used in both architectures. Thus, the total number of block memory used in each

Table 1. Accuracy and distortion performance of SPDFA

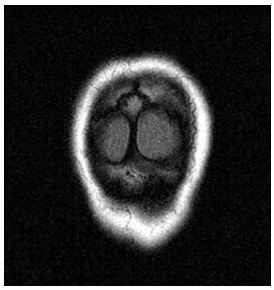
	Reconstructed and original frames						3-D DCT Coefficients		
	PSNR (dB)			RMSE			AvgMaxErr		
Video	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)
MRI1	∞	59	48	≈0	0.28	1.05	0.01	0.18	0.77
MRI2	∞	56	45	≈0	0.40	1.49	0.01	0.23	0.88
Akiyo	∞	58	47	≈0	0.31	1.16	0.01	0.21	0.89
Stefan	∞	56	45	≈0	0.40	1.48	0.01	0.23	0.97
Suzie	∞	56	45	≈0	0.40	1.46	0.01	0.21	0.90
Bus	∞	56	45	≈0	0.40	1.49	0.02	0.23	0.92
Flower	∞	56	45	≈0	0.39	1.45	0.02	0.23	0.97
Mobile	∞	56	45	≈0	0.40	1.49	0.01	0.21	0.92
Average	∞	57	45	≈0	0.37	1.38	0.01	0.22	0.90

Table 2. Accuracy and distortion performance of DPDFA

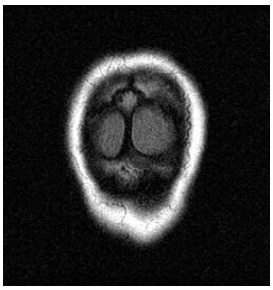
	Reconstructed and original frames						3-D DCT Coefficients		
	PSNR (dB)			RMSE			AvgMaxErr		
Video	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)	(12,8)	(12,4)	(12,2)
MRI1	∞	60	48	≈0	0.25	1.05	0.01	0.18	0.77
MRI2	∞	57	45	≈0	0.36	1.49	0.01	0.23	0.86
Akiyo	∞	59	47	≈0	0.28	1.16	0.01	0.21	0.89
Stefan	∞	57	45	≈0	0.35	1.48	0.02	0.23	0.97
Suzie	∞	57	45	≈0	0.35	1.46	0.01	0.21	0.91
Bus	∞	57	45	≈0	0.35	1.49	0.02	0.23	0.93
Flower	∞	57	45	≈0	0.35	1.45	0.02	0.23	0.97
Mobile	∞	57	45	≈0	0.35	1.40	0.01	0.21	0.92
Average	∞	58	45	≈0	0.33	1.37	0.01	0.22	0.90



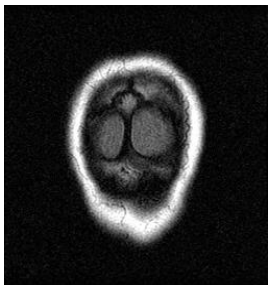
Original MRI2



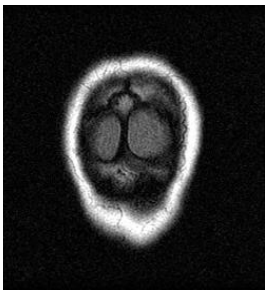
Reconstructed; SPDFA; *(x,y) = (12,2)*



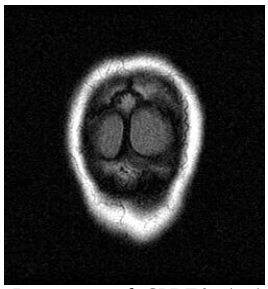
Reconstructed; DPDFA*; (x,y) = (12,2)*



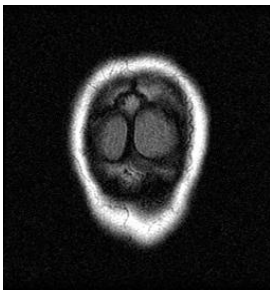
Reconstructed; SPDFA; *(x,y) = (12,4)*



Reconstructed; DPDFA*; (x,y) = (12,4)*



Reconstructed; SPDFA; *(x,y) = (12,8)*



Reconstructed; DPDFA*; (x,y) = (12,8)*

architecture is less than 25% of the available memory resources of the 5vlx50tff1136-3 FPGA device.

Figure 8. The original and reconstructed MRI2 using both Architectures for various wordlength sizes

6.3 Dynamic Power Consumption

The power consumption in FPGA is classified into static and dynamic power. The static power mainly comes from leakage current, whereas charging switch capacitors and short circuit currents are the main sources of dynamic power; hence it can be minimised by switching capacitance reduction [53]. The dynamic power consumption of the presented architectures is shown in Figure 9. The power consumption has been computed using Xilinx Xpower analyser for various clock frequencies and different wordlengths. The dynamic power consumption is higher in DPDFA by around 25-100 mW than SPDFA for selected operating frequencies and wordlengths. The reason behind that is the additional multipliers in the butterfly stages and the duplication of some resources in the first two post addition stages. Thus, SPDFA is outperforming DPDFA in terms of power consumption which makes it a better choice for low power consumption applications.

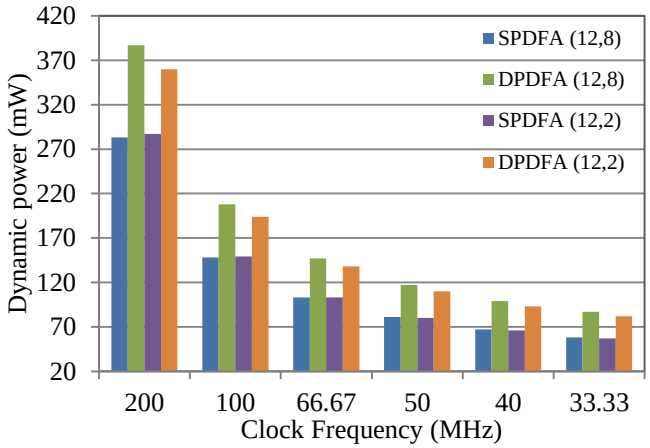


Figure 9: Dynamic power consumption of both architectures.

6.4 Comparison to Similar Work

The throughput of both architectures is 1 coefficient per clock cycle; thus, N^3-clock cycles are needed to compute all the 3-D DCT coefficients of a N^3-word data cube. A comparison between the presented and similar architectures in the literature is shown in Table 4. Of SPDFA and DPDFA, Table 4 shows that the first architecture outperforms the second in terms of area usage as it requires fewer multipliers, adders and registers. The extra hardware DPDFA utilises is needed to perform the dual line computation of the 3-D DCT. Nevertheless, DPDFA is easily pipelined and it has a lower latency and memory requirement than SPDFA. The memory requirements for SPDFA and DPDFA, as listed in Table 4, are used for data reordering and BRO only.

The number of multipliers and adders employed, memory requirements, controller circuits complexity, and computation time of the presented architectures, are also compared to the requirements and performance of the architectures in [38-43]. As shown in Table 4. It can be seen that the presented architectures require the lowest number of multipliers and memory usage of all architectures. Only log_2N and $2log_2N$ multipliers are required to perform the 3-D DCT computation using SPDFA and DPDFA, respectively; for instance, *N* multipliers are required in [41, 42]. In addition, except for the architectures in [43], the presented architectures carry out the 3-D DCT computation with the lowest latency. Table 4 also shows the performance of various architectures in terms of computation time; although the presented architectures exhibit a longer computation time than the work in [39, 43],

this is largely balanced by the presented architectures low hardware usage. This improvement over similar work in the literature is mainly due to the fact that unlike the architectures in [38-43], the focus is on employing and regularising the data flow of a fast algorithm while traditional DCT architectures are based on the direct algorithm [25, 38-43]; this however is not the only benefit of using a VR approach, in fact the control circuits attached to the presented architectures are simple as there is no data transpose. This makes the controller complexity comparable to that of parallel direct approaches in in [38, 42].

7. Conclusions

This paper has presented two new 3-D DCT architectures based on a 3-D DCT VR algorithm. The use of a fast algorithm has yielded architectures with improved processing speed and a reduced hardware usage as they both require the lowest number of arithmetic components and memory requirement among known architectures in the literature; at the same time, such architectures avoid the need for memory transposition and hence are easy to implement and employ a simple control circuitry. The presented architectures are parameterisable in terms of word and transform lengths and exhibit various power consumption, hardware usage, processing speeds and levels of pipelining, which provides the designer with more flexibility and a larger choice when selecting the right architecture for the application under consideration.

References

[1] M. Ayinala and K. K. Parhi, "Parallel pipelined FFT architectures with reduced number of delays," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2012, pp. 63-66.

[2] O. Nibouche, S. Boussakta, M. Darnell, and M. Benaissa, "Algorithms and pipeline architectures for 2-D FFT and FFT-like transforms," *Digital Signal Processing: A Review Journal,* vol. 20, pp. 1072-1086, 2010.

[3] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 20, pp. 1068-1081, 2012.

[4] S. Saponara and B. Neri, "Radar Sensor Signal Acquisition and Multidimensional FFT Processing for Surveillance Applications in Transport Systems," *IEEE Transactions on Instrumentation and Measurement,* vol. 66, pp. 604-615, 2017.

[5] S. Saponara and B. Neri, "Design of compact and low-power X-band Radar for mobility surveillance applications," *Computers & Electrical Engineering,* vol. 56, pp. 46-63, 2016.

[6] A. Das, A. Hazra, and S. Banerjee, "An efficient architecture for 3-D discrete wavelet transform," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 286-296, 2010.

[7] B. K. Mohanty and P. K. Meher, "Memory-efficient architecture for 3-D DWT using overlapped grouping of frames," *IEEE Transactions on Signal Processing,* vol. 59, pp. 5605-5616, 2011.

[8] B. K. Mohanty and P. K. Meher, "Memory efficient modular VLSI architecture for highthroughput and low-latency implementation of multilevel lifting 2-D DWT," *IEEE Transactions on Signal Processing,* vol. 59, pp. 2072-2084, 2011.

[9] S. Al-Azawi, "Low-Power, Low-Area Multi-level 2-D Discrete Wavelet Transform Architecture," *Circuits, Systems, and Signal Processing,* vol. 37, pp. 444-458, 2018.

[10] R. E. Atani, M. Baboli, S. Mirzakuchaki, S. E. Atani, and B. Zamanlooy, "Design and implementation of a 118 MHz 2D DCT processor," in *IEEE International Symposium on Industrial Electronics*, 2008, pp. 1076-1081.

[11] M. Jridi and A. Alfalou, "A low-power, high-speed DCT architecture for image compression: Principle and implementation," in *18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC)*, 2010, pp. 304-309.

[12] M. El Aakif, S. Belkouch, N. Chabini, and M. M. Hassani, "Low power and fast DCT architecture using multiplier-less method," in *2011 Faible Tension Faible Consommation (FTFC)*, 2011, pp. 63-66.

[13] B. Z. Guo, L. Niu, and Z. M. Liu, "Implementation of 2-D DCT based on FPGA," in *Proceedings of SPIE - The International Society for Optical Engineering*, 2010.

[14] G. K. a. S. V. Khurram Bukhari, "DCT and IDCT implementations on different FPGA technologies," *Computer Engineering Lab, Delft University of Technology,* 2009

[15] O. Nibouche, S. Boussakta, and M. Darnell, "Pipeline Architectures for Radix-2 New Mersenne Number Transform," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 56, pp. 1668-1680, 2009.

[16] H. L. P. A. Madanayake, R. J. Cintra, D. Onen, V. S. Dimitrov, and L. T. Bruton, "Algebraic integer based 8×8 2-D DCT architecture for digital video processing," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2011, pp. 1247-1250.

[17] A. M. Shams, A. Chidanandan, W. Pan, and M. A. Bayoumi, "NEDA: A low-power high-performance DCT architecture," *IEEE Transactions on Signal Processing,* vol. 54, pp. 955-964, 2006.

[18] E. D. Kusuma and T. S. Widodo, "FPGA implementation of pipelined 2D-DCT and quantization architecture for JPEG image compression," in *2010 International Symposium in Information Technology (ITSim)*, 2010, pp. 1-6.

[19] S. Al-Azawi, Y. A. Abbas, and R. Jidin, "Low complexity multidimensional CDF 5/3 DWT architecture," in *Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2014 9th International Symposium on*, 2014, pp. 804-808.

[20] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics,* vol. 38, pp. xviii-xxxiv, 1992.

[21] D. J. Le Gall, "The MPEG video compression standard," in *Compcon Spring '91: Digest of Papers*, 1991, pp. 334-335.

[22] W. Li, "Overview of fine granularity scalability in MPEG-4 video standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, pp. 301-317, 2001.

[23] A. Madisetti and A. N. Willson, Jr., "DCT/IDCT processor design for HDTV applications," in *International Symposium on Signals, Systems, and Electronics (ISSSE '95)*, 1995, pp. 63-66.

[24] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, pp. 560-576, 2003.

[25] S. Boussakta and H. O. Alshibami, "Fast algorithm for the 3-D DCT-II," *IEEE Transactions on Signal Processing,* vol. 52, pp. 992-1001, 2004.

[26] X. Li, A. Dick, C. Shen, A. van den Hengel, and H. Wang, "Incremental learning of 3D-DCT compact representations for robust visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35, pp. 863-881, 2013.

[27] S. Sawant and D. A. Adjeroh, "Balanced multiple description coding for 3D DCT video," *IEEE Transactions on Broadcasting,* vol. 57, pp. 765-776, 2011.

[28] H. Y. Huang, C. H. Yang, and W. H. Hsu, "A video watermarking technique based on pseudo-3-D DCT and quantization index modulation," *IEEE Transactions on Information Forensics and Security,* vol. 5, pp. 625-637, 2010.

[29] R. Atta and M. Ghanbari, "Spatio-temporal scalability-based motion-compensated 3-D subband/DCT video coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 16, pp. 43-55, 2006.

[30] S. C. Chan and K. L. Ho, "Direct methods for computing discrete sinusoidal transforms," *IEE Proceedings on Radar and Signal Processing,* vol. 137, pp. 433-442, 1990.

[31] S. An and C. Wang, "Recursive algorithm, architectures and FPGA implementation of the two-dimensional discrete cosine transform," *IET, Image Processing* vol. 2, pp. 286-294, 2008.

[32] G. Jiun-In and L. Chih-Chen, "A generalized architecture for the one-dimensional discrete cosine and sine transforms," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, pp. 874-881, 2001.

[33] Z. Wu, J. Sha, Z. Wang, L. Li, and M. Gao, "An improved scaled DCT architecture," *IEEE Transactions on Consumer Electronics,* vol. 55, pp. 685-689, 2009.

[34] C. Yuan-Ho and C. Tsin-Yuan, "A high performance video transform engine by using space-time scheduling strategy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* vol. 20, pp. 655-664, 2012.
[35] S. Al-Azawi, S. Boussakta, and A. Yakovlev, "High precision and low power DCT architectures for image compression applications," in *IET Conference on Image Processing (IPR)*, 2012, pp. 1-6.
[36] A. Aggoun and I. Jalloh, "Two-dimensional DCT/IDCT architecture," *IEE Proceedings on Computers and Digital Techniques,* vol. 150, pp. 2-10, 2003.
[37] J. I. Guo, "Efficient parallel adder based design for one-dimensional discrete cosine transform," *IEE Proceedings on Circuits, Devices and Systems,* vol. 147, pp. 276-282, 2000.
[38] I. Jalloh, A. Aggoun, and M. McCormick, "3D DCT architecture for compression of integral 3D images," in *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, 2000, pp. 238-244.
[39] A. Aggoun and I. Jalloh, "A parallel 3D DCT architecture for the compression of integral 3D images," in *The 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* 2001, pp. 229-232 vol.1.
[40] M. Bakr and A. E. Salama, "Implementation of 3D-DCT based video Encoder/Decoder system," in *Midwest Symposium on Circuits and Systems*, 2002, pp. II13-II16.
[41] S. Saponara, L. Fanucci, and P. Terreni, "Low-power VLSI architectures for 3D discrete cosine transform (DCT)," in *IEEE 46th Midwest Symposium on Circuits and Systems*, 2003, pp. 1567-1570 Vol. 3.
[42] S. Saponara, "Real-time and low-power processing of 3D direct/inverse discrete cosine transform for low-complexity video codec," *Journal of Real-Time Image Processing,* pp. 1-11, 2012.
[43] Y. Ikegaki, T. Miyazaki, and S. G. Sedukhin, "3D-DCT processor and its FPGA implementation," *IEICE Transactions on Information and Systems,* vol. E94-D, pp. 1409-1418, 2011.
[44] L. Yuanyuan, C. Hexin, Z. Yan, and Y. Chuxi, "Three dimensional DCT similar butterfly algorithm and its pipeline architectures," in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, 2016, pp. 506-510.
[45] S. Al-Azawi, "Efficient Architectures for Multidimensional Discrete Transforms in Image and Video Processing Applications," *PhD Thesis, Newcastle University, UK,* 2013.
[46] L. Yuanyuan, C. Hexin, Z. Yan, and Y. Chuxi, "Device-saving pipeline architectures of multi-dimensional DCT similar butterfly algorithm," in *2016 International Conference on Integrated Circuits and Microsystems (ICICM)*, 2016, pp. 339-344.
[47] J. A. Nikara, J. H. Takala, and J. T. Astola, "Discrete cosine and sine transforms—regular algorithms and pipeline architectures," *Signal Processing,* vol. 86, pp. 230-249, 2006.
[48] O. Alshibami and S. Boussakta, "Fast algorithm for the 3D DCT," in *IEEE International Conference on Acoustics, Speech, and Signal Processing Proceedings (ICASSP '01)*, 2001, pp. 1945-1948 vol.3.
[49] M. C. Lee, R. K. W. Chan, and D. A. Adjeroh, "Fast three-dimensional discrete cosine transform," *SIAM Journal on Scientific Computing,* vol. 30, pp. 3087-3107, 2008.
[50] Xiph.org, "Video Test Media [derf's collection]," *Xiph.org,* [Online]. Available: https://media.xiph.org/video/derf/ Accessed on 20/09/2016.
[51] Q. Huynh-Thu and M. Ghanbari, "The accuracy of PSNR in predicting video quality for different video scenes and frame rates," *Telecommunication Systems,* vol. 49, pp. 35-48, 2012/01/01 2012.
[52] H. R. Sheikh, A. C. Bovik, and G. d. Veciana, "An information fidelity criterion for image quality assessment using natural scene statistics," *IEEE Transactions on Image Processing,* vol. 14, pp. 2117-2128, 2005.
[53] S. McKeown and R. Woods, "Low power field programmable gate array implementation of fast digital signal processing algorithms: Characterisation and manipulation of data locality," *IET Computers and Digital Techniques,* vol. 5, pp. 136-144, 2011.

Saad Al-Azawi received the B.Sc. Degree in Electrical Engineering from University of Baghdad and M.Sc. degree in Electronic and Communication Engineering from Al-Mustansiriya University, Baghdad, Iraq. He received his Ph.D. degree in Electrical and Electronic Engineering from Newcastle University, Newcastle Upon Tyne, England, 2013. Assistant Professor Dr. Saad is currently working as a head of the department of Electronic Engineering, College of Engineering, University of Diyala, Diyala, Iraq. His research interests include Hardware architectures for signal and image processing algorithms and transforms, Digital Image Processing, Digital Signal Processing.

Omar Nibouche received his BEng degree in Electronic Engineering form the Polytechnic School of Algiers and Ph.D. degree in Computer Science from Queen's University Belfast. He is a lecturer in computing in the School of Computing and Mathematics, Ulster University at Jordanstown. His research interests include machine learning, applications of artificial intelligence, computer vision and biometrics.

Said Boussakta received the PhD degree in Electrical Engineering from Newcastle University, U.K., in 1990. Since 1990, he has been working in academia, fully involved in both research and teaching. From 2000-2006 he was at the University of Leeds as a Reader in Digital Communications and Signal Processing. Since 2006, he has been with the School of Engineering, Newcastle University as a Professor of Communications and Signal Processing, lecturing in Communication Networks and Signal Processing subjects. He has supervised over 50 students to Ph.D. completion and published over 200 conference proceedings and journal articles. His research interests are in the areas of fast DSP algorithms, Digital Communications, Communication Network Systems, Cryptography, and Digital Signal/Image Processing. He has also served as Chair in conferences and presented several invited talks in communications, signal processing and security. Prof Boussakta is a Fellow of the IEE, and a Senior Member of the Communications and Signal Processing Societies.

Gaye Lightbody has been a Lecturer within the School of Computing in Ulster University since 2006. She received an M.Eng. (1995) and a PhD (2000) in Electrical and Electronic Engineering from Queen's University of Belfast. Her PhD, High Performance VLSI Architectures for Recursive Least Squares Adaptive Filtering, involved the research and development into a scalable efficient architecture for highly intensive adaptive beamforming applications. Gaye then worked in industry from 2000 to 2006 for Amphion Semiconductor Limited developing intellectual property cores for ASIC and FPGA solutions in the areas of audio, image and video processing. She is a co-author of the book FPGA-based Implementation of Signal Processing Systems published by Wiley in 2008. A second edition has been released in 2017 which provides an update to reflect the latest iterations of

FPGA theory, applications, and technology. This revision includes coverage of FPGA solutions for Big Data Applications.

TABLE 3. Hardware utilisation rate, maximum operating frequencies and computation times for both architectures using various wordlengths

Slice Logic Utilization		Available	SPDFA				DPDFA			
			Hardware usage for wordlength sizes				Hardware usage for wordlength sizes			
			(12,8)	(12,6)	(12,4)	(12,2)	(12,8)	(12,6)	(12,4)	(12,2)
Hardware usage	No of Slice Registers	28,800	1840	1726	1593	1459	3691	3414	3120	2826
	No of Slice LUTs	28,800	2351	2171	1991	1811	3309	3044	2781	2517
	No of occupied Slices	7,200	743	607	588	605	1229	1096	1053	1008
	No of bonded IOBs	480	30	28	26	24	30	28	26	24
	No of 36k BlockRAM used	60	1	-	-	-	1	-	-	-
	No of 18k BlockRAM used		14	15	15	15	15	16	16	16
	No of DSP48Es	48	9	8	8	8	16	12	12	12
Average utilization rate			12%	12%	11%	11%	18%	16%	15%	15%
Maximum frequencies (MHz)			241	230	244	226	266	338	258	333
Computation times for 512×512×8-pixel data (ms)			8.7	9.1	8.6	9.3	7.9	6.2	8.1	6.3

Table 4. Comparison to Similar Architectures in the Literature

Architectures	Adders/Sub.	Multipliers	Memory	Registers	Initial Latency	Computation Time (cycles)	Controller Complexity	DCT Algorithm
[38]	$3N$	$3N$	$N^2(N+1)$	N/R*	N^3+3N	N^3	Simple	Regular; Row-Column-Frame, cascaded
[39]	$\frac{5N^2+N}{2}$	$\frac{5N^2}{2}$	N^3 (transpose memory)	N register between the 2-D DCT and the 1-D-DCT-frame direction	$N^3+\frac{3}{2}N$	N^2	Complex	Regular, Parallel; Row-Column-Frame N×N 1-D DCT+1-D DCT for frame direction
[40]	$3N-3$	$3N$	$N^2(N+1)$	N/R	$>N^2$	$6N^3$ **	Medium	Regular 1-D DCT; Row-Column-Frame
[42]								
Full Parallel (FP)	$N(2N+1)$	$N(2N+1)$ ***	$N(N^2+1)$	N/R	N/R	$2N^2$	Medium	
Cascaded (CS)	$3N$	$3N$ ***	$N^2(N+1)$	N/R	N/R	$2N^3$	Simple	1-D DCT Radix2 Row-Column-Frame
Hardware Multiplexed (HM)	N	N ***	N^3	N/R	N/R	$6N^3$	Complex	
[43]			Input Mem.					
Sequential	N^3	N^3	$2N^3$	$N^3(3N+4)$	N^3	$3N$	Complex	
Pipelined1	$\approx 2N^3$	$2N^3$	$2N^3$	$N^3(3N+4)$	N^3	$3N$	Complex	Regular 1D DCT; Row-Column-Frame
Piplined2	$\approx 3N^3$	$3N^3$	$2N^3$	$N^3(3N+8)$	N^3	$3N$	Complex	
Block	$\frac{N^3}{8}$	$\frac{N^3}{8}$	$2N^3$	$\approx \frac{1}{6}N^3(3N+4)$	N^3	$3N$	Complex	
SPDFA	$12+6log_2N$	log_2N	$\left(\frac{N}{4}+N\right)N^2$ For reordering and BRO	N^3+5N^2+5N+4	$\cong 2N^3$	N^3	Simple	Vector-Radix 3-D DCT
DPDFA	$20+6log_2N$	$2log_2N$	N^3 For reordering and BRO	$\frac{3N^3}{2}+6N^2+11N+8$	$\cong \frac{3}{2}N^3$	N^3	Simple	Vector-Radix 3-D DCT

* N/R : Not reported in their paper.

** Computed for 4×4×4 data block.

*** Multiplication is performed by a serial distributed arithmetic architecture.